

IAB207 Group17: Assignment 2 Declaration

Section 1: Team declaration (functional / non-functional)

Please declare your team details on this submission itself by completing and including one of the two table options below. The details should reflect the same team as defined in your canvas group. It will assist the marker.

Option A: – Most will use this option. Delete if not used

Functional Team – most teams should use this one		
Canvas team identifier:	Group17	
Tutor:	Alaa	
Day and time of tutorial:	Monday, 2:00 -4:00	
GitHub name and link:	https://github.com/Jairobuifranco/A2_Group17	
Full name	Student#	QUT Email
Jairo Buitrago	n11379146	N11379146@qut.edu.au
Liang Feiyu	n12155551	N12155551@qut.edu.au
Jonty Kirkpatrick	n12020958	N12020958@qut.edu.au
Issac Habtemichael	n11911123	N11911123@qut.edu.au

Backlog:

<https://qut.padlet.org/jontykirkpatrick/iab207-scrum-board-team-17-6qqztdasq8lp7azs>

Section 2: Team contribution statement

Brief overview from each team member explaining how they contributed: **State your role within the team and the tasks that you completed**. There is no word limit, however, 60-100 words should be sufficient.

Jairo Alberto Buitrago Franco (n11379146)	Home page, Database, application architecture and skeleton, scrum master.
Liang Feiyu (n12155551)	Implement Create/Update page
Jonty Kirkpatrick (n12020958)	Login page, Sign up, user authentication
Issac Habtemichael (n11911123)	Booking page

Section 3: git shortlog > output.txt

Paste git shortlog output here. The git shortlog command provides a summarised view of the git log output, useful for release announcements and reporting. It groups commits by author and displays the first line of each commit message, offering a quick overview of contributors' work. This will help the marker confirm who has worked on what.

```
11 FayeL11 <n1215555@qut.edu.au>
8 FayeL11 <n12155551@qut.edu.au>
7 JontySteele <jonty.kirkpatrick@connect.qut.edu.au>
6 "Bigbird101-101" <"endhigh33@gmail.com">
4 Jairo Alberto Buitrago Franco <jairoalbertobuitragofranco@MacBookPro.tpg_fwa>
3 Jairo Alberto Buitrago Franco <jairoalbertobuitragofranco@Jairos-MacBook-
Pro.local>
1 Bruh <endhigh33@gmail.com>
```

Section 4: Team use of AI statement

Instructions:

- **We did not use AI.** Include part A. Delete part B.
- **We (one or more team members) used AI in an authorized way** (as described in the assignment brief). Complete part B. Delete part A.
- **We are not sure...** please speak to your tutor before submitting the assignment.

Part B:

As a team, we report that some members have used AI in the authorized way stated in the (Use of Artificial Intelligence (AI) in this unit) page on canvas.

Summary of AI use (Liang Feiyu, 12155551)

Prompt1:

Are my pages "convincing, comprehensive, and robust"?

HTML Page Review and Improvement Suggestions



Are my pages "convincing, comprehensive, and robust"?

Looking at your two HTML pages, I'd say they are **quite convincing, comprehensive, and robust** overall. Here's my detailed assessment:

Strengths (What makes them convincing and robust):

Visual Design & User Experience

- Clean, professional Bootstrap styling
- Consistent branding and navigation
- Responsive design that works on different screen sizes
- Good use of icons and visual hierarchy
- Appropriate color coding for status badges

Functionality & Features

- **MyEvents.html:** Comprehensive event management with status handling, conditional buttons, and proper validation
- **Create.html:** Rich form with both file upload and URL options for images
- Good error handling and user feedback
- Conditional rendering based on event status and timing



Technical Robustness

- Proper form validation with error display
- Security measures (CSRF protection with `form.hidden_tag()`)
- Image handling with fallbacks and multiple sources
- Date/time validation logic
- Accessibility considerations (aria labels)

⚠ Areas for Improvement:

1. Navigation Consistency

In `create.html`, you have two "active" nav links simultaneously:

⚠ Areas for Improvement:

1. Navigation Consistency

In `create.html`, you have two "active" nav links simultaneously:

```
html
Copy Download Run

<li class="nav-item"><a class="nav-link active" href="{{ url_for('events.create_event') }}" aria-current="page">Create Event</a></li>
<li class="nav-item"><a class="nav-link active" href="{{ url_for('events.my_events') }}" aria-current="page">My Events</a></li>
```

Only the current page should have the `active` class.

2. Form Submission Logic

In `create.html`, you have conflicting submit buttons:

```
html
Copy Download Run

{{ form.submit(class="btn btn-primary") }}

{%- if action == 'Update Event' %}
<button type="submit" class="btn btn-warning">
  <i class="bi bi-arrow-clockwise"></i> Update Event</button>
{%- endif %}
```

You should use the form's submit field consistently.

3. Error State Styling

Add the `is-invalid` class to form fields when there are errors:

```
html Copy Download Run  
{% form.name(class="form-control" + (" is-invalid" if form.name.errors else "")) %}
```

4. Image Path Handling

Your image path logic could be simplified:

```
html Copy Download Run  
<!-- Instead of this complex logic -->  
{% if event.image_url.startswith('http') %}  
  
{% else %}  
  
{% endif %}  
  
<!-- Consider standardizing how you store image paths -->
```

5. Missing Empty States

- Consider adding loading states
- Add "no results" messaging for filtered views

6. Accessibility

- Add more `alt` text descriptions
- Ensure color contrast meets WCAG guidelines
- Add `label` associations for all form fields

Overall Rating: 8.5/10

Your pages are **definitely convincing** - they look professional and trustworthy. They're **comprehensive** - covering most user needs and edge cases. They're **robust** - with good error handling and validation.

Final verdict: Yes, these are production-quality pages that would work well in a real application. The issues noted are relatively minor and could be addressed with some polishing.

The attention to detail in handling different event states, conditional button enabling/disabling, and

{% else %}

```

<div class="text-center py-5">
  <i class="bi bi-calendar-x text-muted" style="font-size: 4rem;"></i>
  <h4 class="mt-3">You haven't created any events yet</h4>
  <a href="{{ url_for('main.create_event') }}" class="btn btn-primary">Create Your First
Event with LocalConcerts</a>
</div>
{% endif %}

```

Code adding for advice 5 and 6, remind users have not created any events yet

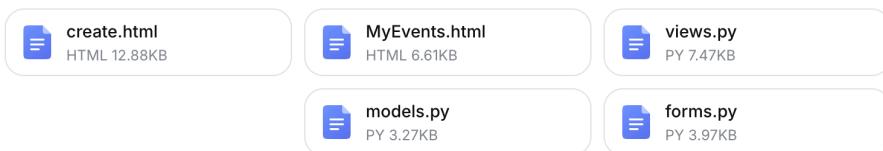
===== NEXT SAMPLE OF AI USE =====

}

Prompt2:

How can I improve my code to meet the requirement: In some sort of standalone page like "My Events", accessible only to logged-in users, a user can see all events they have created and choose one to update/cancel. Note that you could alternatively have this live on the Booking History page, or some other sensible location.

Enhancing My Events Page Functionality and UX



How can I improve my code to meet the requirement: In some sort of standalone page like "My Events", accessible only to logged-in users, a user can see all events they have created and choose one to update/cancel. Note that you could alternatively have this live on the Booking History page, or some other sensible location.

Looking at your code, I can see you've implemented most of the "My Events" functionality well. Here are some improvements to make it more robust and user-friendly:

1. Fix Navigation Active States

In `create.html`, fix the active state for "My Events" link:

```

html
Copy Download | Run

<!-- In create.html navbar -->
<li class="nav-item"><a class="nav-link" href="{{ url_for('events.my_events') }}>My Events
</a></li>
<li class="nav-item"><a class="nav-link active" href="{{ url_for('events.create_event') }}"
aria-current="page">Create Event</a></li>

```

The actual code is:

`href={{ url_for('main.create_event') }}` to make it simple, link to main page

2. Improve Event Status Handling

In `models.py`, enhance the status logic:

```
python
Copy Download

class Event(db.Model):
    # ... existing fields ...

    @property
    def can_be_modified(self) -> bool:
        """Check if event can be updated or cancelled."""
        if self.status != 'Open':
            return False
        if self.start_time < datetime.utcnow():
            return False
        return True

    @property
    def available_tickets(self) -> int:
        """Calculate available tickets."""
        total_ordered = sum(order.quantity for order in self.orders)
        return max(0, self.ticket_quantity - total_ordered)
```

3. Enhanced My Events Template

Improve `MyEvents.html` with better information and controls:

```
html
Copy Download | Run

<!-- Add this after the page title -->
<div class="d-flex justify-content-between align-items-center mb-4">
    <div>
        <h3 class="mb-1">My Created Events</h3>
        <p class="text-muted mb-0">Manage your events and track their status</p>
    </div>
    <a href="{{ url_for('events.create_event') }}" class="btn btn-primary">
        <i class="bi bi-plus-circle me-1"></i> Create New Event
    </a>
</div>

<!-- Add statistics summary -->
<div class="row mb-4">
    <div class="col-md-3">
        <div class="card bg-primary text-white">
            <div class="card-body text-center">
                <h4>{{ events|length }}</h4>
```

I added a welcome message “Welcome to Local Concerts” and a subheading “Filled out the form to create exciting events”

4. Enhanced Event Cards with More Info

In `MyEvents.html`, enhance the event cards:

```
html
Copy Download Run

<div class="card-body">
  <div class="d-flex justify-content-between align-items-start mb-2">
    <h5 class="card-title">{{ event.title }}</h5>
    {% if event.status == 'Open' %}
      <span class="badge bg-success">{{ event.status }}</span>
    {% elif event.status == 'Cancelled' %}
      <span class="badge bg-danger">{{ event.status }}</span>
    {% else %}
      <span class="badge bg-secondary">{{ event.status }}</span>
    {% endif %}
  </div>

  <p class="card-text mb-1">
    <strong><i class="bi bi-geo-alt"></i> Venue:</strong> {{ event.venue }}</p>
  <p class="card-text mb-1">
    <strong><i class="bi bi-calendar-event"></i> Date:</strong>
```

5. Add View Event Details Button

In the card footer, add a view details button:

```
html
Copy Download Run

<div class="card-footer bg-transparent">
  <div class="d-flex justify-content-between align-items-center">
    <!-- View Details Button -->
    <a href="{{ url_for('main.event', event_id=event.id) }}">
      <span class="btn btn-outline-info btn-sm">
        <i class="bi bi-eye"></i> View
      </span>
    </a>

    <div class="btn-group">
      {% if event.status == 'Open' and event.start_time > now %}
        <a href="{{ url_for('events.update_event', event_id=event.id) }}">
          <span class="btn btn-outline-primary btn-sm">
            <i class="bi bi-pencil"></i> Update
          </span>
        </a>
        <form action="{{ url_for('events.cancel_event', event_id=event.id) }}" method="post">
          <span class="d-inline">
            <button type="submit" class="btn btn-outline-danger btn-sm">
              <span>Delete</span>
            </button>
          </span>
        </form>
      {% else %}
        <span>Event Details</span>
      {% endif %}
    </div>
  </div>
</div>
```

6. Add Sorting and Filtering Options

Add sorting controls to `MyEvents.html`:

```
html Copy Download Run  
  
<!-- Add this after the statistics cards --&gt;<div class="card mb-4">

Filter by Status:

AllActiveCancelled



Sort by:


```

7. Enhanced My Events View with Filtering

Update `views.py` to handle filtering and sorting:

```
python Copy Download  
  
@events_bp.route('/my-events')  
@login_required  
def my_events():  
    # Base query  
    query = Event.query.filter_by(user_id=current_user.id)  
  
    # Filter by status  
    status_filter = request.args.get('status')  
    if status_filter in ['Open', 'Cancelled']:  
        query = query.filter(Event.status == status_filter)  
  
    # Sorting  
    sort_by = request.args.get('sort', 'date_desc')  
    if sort_by == 'date_asc':  
        query = query.order_by(Event.start_time.asc())  
    else: # date_desc  
        query = query.order_by(Event.start_time.desc())
```

8. Add Empty State with Better Guidance

Enhance the empty state in `MyEvents.html`:

```
html Copy Download Run  
  
{% else %}  
<div class="text-center py-5">  
  <i class="bi bi-calendar-x text-muted" style="font-size: 4rem;"></i>  
  <h4 class="mt-3">You haven't created any events yet</h4>  
  <p class="text-muted mb-4">Start by creating your first event to share with the community</p>  
</div>  
<div class="d-flex justify-content-center gap-3">  
  <a href="{{ url_for('events.create_event') }}" class="btn btn-primary btn-lg">  
    <i class="bi bi-plus-circle me-2"></i>Create Your First Event  
  </a>  
  <a href="{{ url_for('main.index') }}" class="btn btn-outline-secondary btn-lg">  
    <i class="bi bi-house me-2"></i>Browse Events  
  </a>  
</div>  
{% endif %}
```

9. Add Success Messages for Better UX

In your view functions, add more specific flash messages:

```
python Copy Download  
  
@events_bp.route('/cancel/<int:event_id>', methods=['POST'])  
@login_required  
def cancel_event(event_id):  
    event = db.session.get(Event, event_id)  
    if event is None:  
        flash('Event not found.', 'error')  
        return redirect(url_for('events.my_events'))  
  
    if event.user_id != current_user.id:  
        flash('You are not authorized to cancel this event.', 'warning')  
        return redirect(url_for('events.my_events'))  
  
    try:  
        event.status = 'Cancelled'  
        db.session.commit()  
        flash(f'Event "{event.title}" has been cancelled successfully.', 'success')  
    except Exception as e:  
        db.session.rollback()
```

===== NEXT SAMPLE OF AI USE =====

Prompt:

(e) Create / update event:

If a user tries to create/update an event but is not logged in, they should be redirected to the login view.

Have you ensured all relevant event attributes are collected upon creation?

The event status should NOT be selectable on event creation. All new events should be set to "Open". Similarly, when updating an event, an event can be cancelled but the status cannot manually change.

Input validation (through Flask Form validators) is crucial here. For example - have you made sure ticket quantities and ticket prices cannot be negative? Have you ensured event date must be in the future? Ensure to make use of InputRequired where appropriate too.

For updating an event, it's usually a good idea to prefill the form with the pre-populated existing event details. That makes it faster for the creator to quickly make a change, for example if they made a spelling mistake.

(e) Create / update event:

If a user tries to create/update an event but is not logged in, they should be redirected to the login view.

Have you ensured all relevant event attributes are collected upon creation?

The event status should NOT be selectable on event creation. All new events should be set to "Open". Similarly, when updating an event, an event can be cancelled but the status cannot manually change.

Input validation (through Flask Form validators) is crucial here. For example - have you made sure ticket quantities and ticket prices cannot be negative? Have you ensured event date must be in the future? Ensure to make use of InputRequired where appropriate too.

For updating an event, it's usually a good idea to prefill the form with the pre-populated existing event details. That makes it faster for the creator to quickly make a change, for example if they made a spelling mistake.

Looking at your code, I can see you've implemented most of these requirements well, but there are some areas that need improvement. Here's a comprehensive review and fixes:

1. Authentication Check for Create/Update Events

Your current implementation already has `@login_required` decorators, which is correct. However, let's ensure the redirect works properly by adding this to your `views.py`:

```
python Copy  Download   
  
from flask import redirect, url_for  
from flask_login import current_user  
  
# Your existing decorators are correct:  
@events_bp.route('/create', methods=['GET', 'POST'])  
@login_required # This automatically redirects to login if not authenticated  
def create_event():  
    # ... existing code ...  
  
@events_bp.route('/update/<int:event_id>', methods=['GET', 'POST'])  
@login_required # This automatically redirects to login if not authenticated  
def update_event(event_id):  
    # ... existing code ...
```

2. Remove Status Field from Forms

Your current implementation correctly **does not** have a status field in the form, and new events are set to "Open" by default. This is correct.

3. Improve Input Validation in Forms

Your `forms.py` has good validators, but let's enhance them:

```
python Copy  Download   
  
class EventForm(FlaskForm):  
    # ... existing fields ...  
  
    # Enhanced validators  
    name = StringField('Event Name', validators=[  
        InputRequired('Please enter an event name'),  
        Length(min=2, max=150, message='Event name must be between 2 and 150 characters')  
    ])  
  
    venue = StringField('Venue', validators=[  
        InputRequired('Please enter a venue'),  
        Length(min=2, max=150, message='Venue must be between 2 and 150 characters')  
    ]) ▼
```

4. Fix Form Pre-population in Update View

Your current pre-population logic is good, but let's make it more robust in `views.py`:

```
python Copy  Download   
  
@events_bp.route('/update/<int:event_id>', methods=['GET', 'POST'])  
@login_required  
def update_event(event_id):  
    event = db.session.get(Event, event_id)  
    if event is None:  
        abort(404)  
  
    # Enhanced authorization check  
    if event.user_id != current_user.id:  
        flash('You are not authorized to update this event.', 'warning')  
        return redirect(url_for('events.my_events'))  
  
    # Check if event can be modified  
    if not event.can_be_modified:  
        flash('This event cannot be modified because it has already started or is cancelled.', 'warning')  
        return redirect(url_for('events.my_events'))  
  
    form = EventForm()
```

5. Enhanced Create Event View

Improve the create event function in `views.py`:

```
python Copy  Download   
  
@events_bp.route('/create', methods=['GET', 'POST'])  
@login_required  
def create_event():  
    form = EventForm()  
  
    if form.validate_on_submit():  
        try:  
            # Combine date and time  
            start_datetime = datetime.combine(form.date.data, form.start_time.data)  
            end_datetime = datetime.combine(form.date.data, form.end_time.data)  
  
            # Image handling  
            image_url_to_save = None  
            if form.image_file.data and form.image_file.data.filename:  
                image_url_to_save = save_uploaded_file(form.image_file.data)  
  
            if not image_url_to_save and form.image_url.data and form.image_url.data.strip()  
        .
```

6. Add Helper Method to Event Model

Add this to your `models.py` to help with modification checks:

python

Copy  Download

```
class Event(db.Model):
    # ... existing fields ...

    @property
    def can_be_modified(self) -> bool:
        """Check if event can be updated (not cancelled and in future)"""
        if self.status == 'Cancelled':
            return False
        if self.start_time < datetime.utcnow():
            return False
        return True
```

7. Fix Template Issues

In your `create.html`, fix the duplicate submit button:

html

Copy  Download | 

```
<!-- Remove the duplicate submit button in the action buttons section -->


{% if action == 'Update Event' %}
        <button type="submit" class="btn btn-success">
            <i class="bi bi-pencil-square"></i> Update Event
        </button>
    {% else %}
        <button type="submit" class="btn btn-primary">
            <i class="bi bi-plus-circle"></i> Create Event
        </button>
    {% endif %}


<a class="btn btn-outline-secondary" href="{{ url_for('events.my_events') }}>
    <i class="bi bi-calendar-event"></i> My Events
</a>
</div>
```

Summary of Key Improvements:

1. **Authentication:** Already handled by `@login_required`
2. **Status Management:** Status is automatically set to "Open" and cannot be manually changed
3. **Input Validation:** Enhanced validators for all fields
4. **Form Pre-population:** Improved logic for update forms
5. **Error Handling:** Better flash messages and error handling
6. **Security:** Authorization checks for event ownership

Your implementation is already quite good - these changes will make it more robust and user-friendly while meeting all the specified requirements.